Validation of Mixed SIGNAL-ALPHA Real-Time Systems through Affine Calculus on Clock Synchronisation Constraints

Irina M. Smarandache[1], Thierry Gautier[2], and Paul Le Guernic[2]

[1] The University of Reading, Department of Computer Science
Whiteknights, PO Box 225, Reading RG6 6AY, United Kingdom
Tel.: (44) 118 931 8611 (7626), Fax: (44) 118 975 1994
I.M.Smarandache@reading.ac.uk
[2] IRISA-INRIA, Campus de Beaulieu, 35042 Rennes Cedex, France
Thierry.Gautier@irisa.fr, Paul.LeGuernic@irisa.fr

Abstract. In this paper we present the affine clock calculus as an extension of the formal verification techniques provided by the SIGNAL language. A SIGNAL program describes a system of clock synchronisation constraints the consistency of which is verified by compilation (clock calculus). Well-adapted in control-based system design, the clock calculus has to be extended in order to enable the validation of SIGNAL-ALPHA applications which usually contain important numerical calculations. The new affine clock calculus is based on the properties of affine relations induced between clocks by the refinement of SIGNAL-ALPHA specifications in a codesign context. Affine relations enable the derivation of a new set of synchronisability rules which represent conditions against which synchronisation constraints on clocks can be assessed. Properties of affine relations and synchronisability rules are derived in the semantical model of traces of SIGNAL. A prototype implementing a subset of the synchronisability rules has been integrated in the SIGNAL compiler and used for the validation of a video image coding application specified using SIGNAL and ALPHA.

1 Introduction

Real-time systems, and more generally reactive systems [4], are in continuous interaction with their environment. Therefore, they must respond *in time* to external stimuli. Moreover, real-time systems must be safe, thus one would wish to prove their correctness. Time constraints and safety are two important aspects to be considered in the design of a real-time application.

Real-time systems may be constrained by very tight real-time deadlines. Moreover, a hardware implementation of parts of these systems is sometimes required, to meet specific constraints for instance. An example is an application consisting of numerical calculations performed iteratively on large structures of regular multidimensional data. In this case, a hardware/software implementation may be envisaged, in which the numerical calculations are conveyed to hardware

for efficiency reasons, while the control relating these parts is implemented in software.

In general, designing a mixed hardware/software real-time system requires a rigorous methodology that comprises methods and tools addressing, among others, system specification and validation, optimal code generation and hardware synthesis. These aspects are dealt with in *codesign* [7] [9] which denotes the specification, validation and implementation of an application which consists both of a hardware part, in the form of a set of specialised integrated circuits, and a software part implemented on general programmable processors. The idea is to explore various possible implementations of hardware/software systems in order to improve their performance and to ensure the respect of cost constraints.

1.1 Real-Time System Codesign

System codesign is a complex process which can be decomposed into three main activities [7]: 1. The *cospecification* of an application at various levels of abstraction; 2. The validation of a specification by formal verification or simulation, also known as *cosimulation*; 3. The hardware/software partitioning of an application, the evaluation of a partitioning from the point of view of the time constraints and cost, the generation of executable code, the synthesis of hardware, and the production of the interface between hardware and software, i.e *cosynthesis*. A lot of work has been done, the purpose of which was to define a well-structured methodology for codesign [7] [11] [19]. An important point was generally the description of both hardware and software using the same language, like for instance VHDL enhanced with mechanisms for calling C functions [14], or high-level languages like C, C++ or FORTRAN extended with facilities for the description of hardware systems [10]. These approaches enable the programming of both the hardware and software parts of a system in a unique framework and their validation by simulation. However, they cannot guarantee system correctness. This aspect can be much improved by using formal languages for system specification, refinement of specifications towards lower levels of abstraction (implementation) and validation of the various specifications by formal verification.

Defining a complete methodology of codesign requires addressing other relevant problems, most of them concerning cosynthesis. Among these problems there are the automatic partitioning into hardware and software, the synthesis of hardware and the generation of optimal code for software implementation.

The work presented in this paper is part of a more general effort for building a hybrid framework in which the SIGNAL [12] [13] and ALPHA [20] languages can be used for real-time system codesign.

1.2 Cospecification and Cosimulation of SIGNAL-ALPHA Systems

SIGNAL is a *synchronous* [4] language developed for the specification, validation and implementation of real-time systems. SIGNAL variables represent finite or infinite sequences of values (data) which can be filtered or merged before being submitted to classical boolean or mathematical operations. A *clock* is implicitly

associated with each SIGNAL variable: it represents a set of temporal indices which denote the logical instants where the variable is present and has a value. The semantics of a SIGNAL program can be described by a system of constraints (relations) on clocks and values, which is constructed and verified for consistency during compilation. The verification of the clock constraints is called *clock calculus*. The SIGNAL environment is enhanced with tools for C [5] and VHDL [3] code generation and formal verification of dynamic properties [2].

In its present form, SIGNAL is well-adapted for the design of control-based real-time systems. Firstly, this is due to its limitations concerning the treatment of computations on multidimensional data such as matrices. Only simple algorithms can be expressed in SIGNAL and no significant optimisation is performed at the level of the generation of executable C or VHDL code concerning vectors. In contrast with SIGNAL, the ALPHA language has been developed primarily for the specification and implementation of algorithms on multidimensional data. Such algorithms can be described in ALPHA using affine recurrence equations over convex polyhedral domains [20] and be further transformed for optimal hardware or software implementation on parallel or sequential architectures [21].

Given their complementary properties, the SIGNAL and ALPHA languages can be used jointly for the design of real-time systems containing important numerical calculations on multidimensional data and control: numerical computations are expressed in ALPHA and the control is conveyed to SIGNAL. When the real-time requirements of the system are very tight, a mixed hardware/software implementation may be envisaged. In [9] we propose a hybrid framework for the combined use of SIGNAL and ALPHA in real-time system codesign. In order for this framework to be operational, it is necessary to interface SIGNAL and ALPHA programs both at the functional and architectural level. The former corresponds to a high-level mathematical representation of an algorithm in ALPHA, while the latter contains a set of new temporal indices corresponding to the execution of the algorithm on a parallel or sequential architecture.

In SIGNAL-ALPHA systems, the refinement of an ALPHA program from a functional level to an architectural level oriented toward a particular implementation also induces a refinement of the temporal indices in SIGNAL. The new time indices are obtained through *affine transformations* on the instants of time of the initial SIGNAL specification. Consider clocks c and c_1 in SIGNAL which are identical at the functional level (they are also denoted as *synchronous*). After refinement, their relative position is such that clock c_1 can be obtained by an *affine transformation* applied to clock c: the instants of time of c and c_1, denoted respectively T and T_1, can be described by a pair of *affine functions* $T = \{nt + \varphi_1 \mid t \in \mathcal{T}\}$, $T_1 = \{dt + \varphi_2 \mid t \in \mathcal{T}\}$, on the same set of instants $\mathcal{T}$. With $\varphi = \varphi_2 - \varphi_1$, we will say that clock c_1 is obtained by an (n, φ, d)-affine transformation applied to clock c, where $n, d \in \mathbb{N}^*$ the set of strictly positive integers and $\varphi \in \mathbb{Z}$ the set of integers. Clocks c and c_1 are also said to be in an (n, φ, d)-*affine relation*.

Clocks obtained by affine transformation may be re-synchronised at the architectural level. As an example, consider clocks c, c_1 and c_2 which are identical

in the SIGNAL functional specification. At the architectural level, clocks c_1 and c_2 have been transformed such that c, c_1 and c, c_2 are respectively in affine relations of parameters (n_1, φ_1, d_1) and (n_2, φ_2, d_2). Whether clocks c_1 and c_2 can be re-synchronised depends on the properties of the affine relations which are induced from the values of (n_1, φ_1, d_1) and (n_2, φ_2, d_2). Moreover, the relations between c, c_1 and respectively, c, c_2 may be expressions on (n, φ, d)-affine relations constructed using operations like composition, union, etc. In this case, the re-synchronisation of clocks c_1 and c_2 depends on the properties of these operations.

The SIGNAL clock calculus performs the verification of clock synchronisation constraints using a set of *synchronisability rules*, i.e. conditions against which these constraints can be assessed. The current clock calculus depends on boolean equation resolution methods [5] [1] which have been successfully used for the validation of numerous control-based real-time applications. However, in order to validate mixed SIGNAL-ALPHA systems as presented above, it is necessary to extend the current clock calculus with a set of synchronisability rules deduced from the properties of (n, φ, d)-affine relations. The new set of rules defines the *affine* clock calculus, which constitutes the main topic of this paper. We explore the space of (n, φ, d)-affine relations and study to which extent it is closed under the main operations that can be performed on affine relations. Following this study, we define a set of synchronisability rules which, although incomplete, enables the validation of the principles underlying the cospecification and cosimulation using SIGNAL and ALPHA. The semantical model of traces of SIGNAL [12] [16] constitutes the support for the study of the properties of affine relations and for the definition of the new synchronisability rules.

1.3 Organisation of the Paper

In Section 2 we present the integration of SIGNAL and ALPHA for system codesign. Section 3 is the central core of this paper and is dedicated to the definition and implementation of the affine clock calculus. The main concepts useful for this purpose are progressively introduced: these are the model of traces of the SIGNAL language, the properties of affine relations on clocks, the set of synchronisability rules induced by the latter, and finally the necessary elements for the integration of the affine clock calculus in the compiler. The affine clock calculus has been applied to the cospecification and cosimulation of a video image coding application; this is briefly illustrated in Section 4. In the same section we discuss in which way the SIGNAL and ALPHA environments may further contribute to the development of a complete codesign methodology based on both languages. Finally, in Section 5 we present conclusions and perspectives of our work.

2 SIGNAL and ALPHA in Real-Time System Codesign

Figure 1 summarizes the main elements of the environments around SIGNAL and ALPHA that make both languages well-adapted for real-time system codesign.

SIGNAL and ALPHA programs represent mathematical notations for the properties of the processes they define. The system of constraints on clocks and values associated with a SIGNAL program is transformed by compilation into a *synchronised data flow graph* (SDFG). This data structure constitutes the support for executable code generation (C or VHDL) or verification of dynamic properties using the formal tool SIGALI [2].

The ALPHA compiler includes a powerful type checking mechanism based on the structure of an ALPHA variable as a function over convex polyhedra. The syntax tree obtained after compilation can be directly translated into C code for functional simulation, or it can be transformed into a subset of ALPHA called ALPHA0 which exhibits the details of a parallel or sequential implementation. The syntax tree in ALPHA0 form can be further translated in C or VHDL executable code or directly mapped on a netlist [21].

The interface between SIGNAL and ALPHA is based on the fact that both languages can be translated in C and executed for functional simulation. Furthermore, SIGNAL offers the possibility to call *external processes*: such a process can be the specification of an algorithm in a language other than SIGNAL. A particular type of an external process is a *function*, the execution of which is considered instantaneous from the point of view of SIGNAL. A SIGNAL function can be a predefined or a user-defined C function.

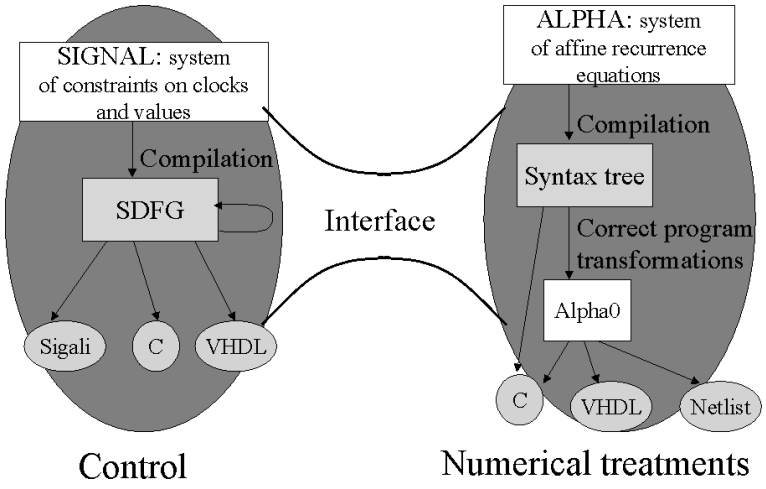

Fig. 1. SIGNAL and ALPHA in system codesign.

2.1 Functional Cospecification and Cosimulation

Being a synchronous language, SIGNAL is based on the following hypotheses [4]:
1. All actions (communications and calculations) in a system have zero *logical*

duration (the elapsed time is represented by the precedence of successive values on a same data flow); 2. Two or more actions can take place at the same logical instant, such actions being termed "simultaneous". From the point of view of the logical temporal properties of a system, only succession and simultaneity of instants are of interest. Although their exact time values are not considered, note however that they will be considered for a given implementation. The process associated with a SIGNAL program represents thus a succession of logical instants, with each instant being associated one or more actions considered of zero logical duration and involving process variables present at that instant.

Consider for example a coding system for sequences of video images at 34 Mbits/s [8]. A system of this type consists of a set of numerical treatments applied iteratively on images of the same dimension. Images are divided into luminance and chrominance blocks and treatments are applied to each block. Numerical treatments consist mainly of algorithms for inter and intra image coding which require operations like a discrete cosine transformation (DCT). In order to illustrate the interfacing between SIGNAL and ALPHA, we have isolated from the coding application a simple SIGNAL program and have illustrated the associated process in Fig. 2. It consists of a DCT operation applied in sequence to different values A_i of the matrix of pixels A present at each logical instant of time t_i. The matrix A corresponds to a block of luminance or chrominance of an image. The DCT can be expressed in SIGNAL as $B := Dct(A)$, where DCT is actually an external process. The DCT is a time consuming algorithm, particularly for large matrices or when applied to images containing a large number of blocks. In order to improve the overall performance of the coding application, one would wish to execute each instance $B_i := Dct(A_i)$ on a parallel integrated architecture as derived by the ALPHA environment.

The DCT can be easily described in ALPHA. The SIGNAL-ALPHA cospecification and cosimulation of the new system is made possible at the *functional* level as follows (see Fig. 2): 1. The ALPHA *system* is translated in executable C code; 2. The C function *ALPHA_C* obtained at step 1 represents the external process implementing the DCT in SIGNAL. The function *ALPHA_C* is considered instantaneous in SIGNAL; the clocks of the matrices A and B, denoted respectively by c and c_1, are therefore synchronous. The overall system is thus represented as a SIGNAL specification executing instantaneously the functional description of the ALPHA specification. The system can be validated in the SIGNAL environment by formal verification (compilation, model checking with SIGALI) and/or simulation.

2.2 Implementation-oriented Cospecification and Cosimulation

A mixed SIGNAL-ALPHA specification at the functional level may be refined in order to take into consideration the details of a particular implementation. The ALPHA program of Section 2.1 describing a DCT may be submitted to a sequence of transformations for a parallel or sequential implementation. These transformations guarantee the equivalence of the final specification, noted *ALPHA'* in Fig. 3, with the initial *ALPHA* system of Fig. 2. The system *ALPHA'* contains

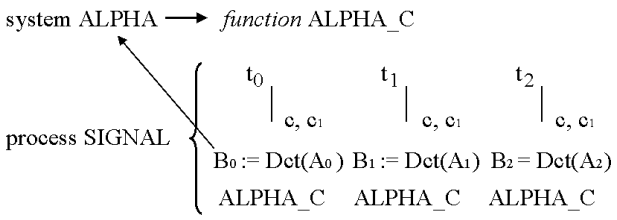

Fig. 2. SIGNAL-ALPHA interface at functional level.

the time indices corresponding to a particular scheduling of the DCT operation. In Fig. 3 these time indices are represented as the diagonal sets of *micro*-instants μt_i^j associated with each *macro*-instant t_i.

The SIGNAL specification has to be refined accordingly in order to enable the validation of the overall system. Therefore, the micro-instants of time of *ALPHA'* are taken into consideration in the new process *SIGNAL'* and described as the sets of instants μSt_0^i, μSt_1^i, etc. (see Fig. 3). The C function *ALPHA'_C* has been derived from *ALPHA'* and transformed in order to describe the sequence of operations performed at each micro-instant of time.

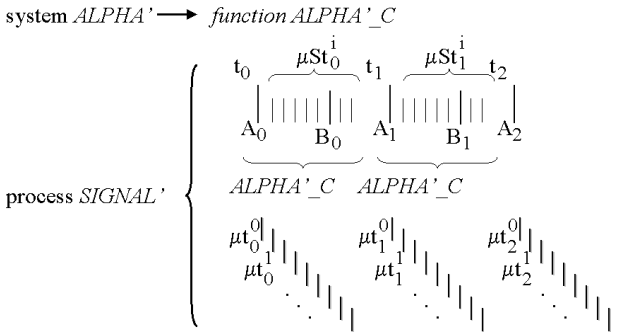

Fig. 3. SIGNAL-ALPHA interface at architectural level.

The regularity of ALPHA values manifests itself in SIGNAL in several ways. First, the sets of micro-instants μSt_0^i, μSt_1^i, etc. have the same cardinality. Also, successive values for B are provided at specific micro-instants between any two successive macro-instants t_i and t_{i+1} in a regular manner. This situation is illustrated in Fig. 4 where the clocks of matrices A and B, denoted respectively by c and c_1, are defined by the following instants of time: $c = \{0, 9, 18, ...\}$ and $c_1 = \{6, 15, ...\}$ (after providing the values B_i at the instants of time defined by c_1, the architecture implementing the operation $B_i := Dct(A_i)$ may execute further computations like initialisations for the next operation $B_{i+1} := Dct(A_{i+1})$).

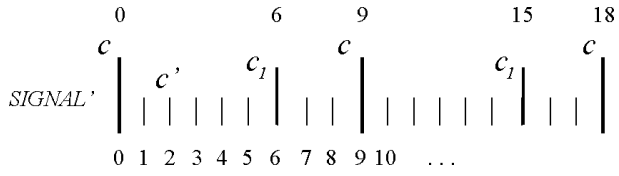

Fig. 4. Illustration of an affine relation.

In Fig. 4, clock c' is defined by the set of instants $\{0, 1, 2, 3, 4, 5, ...\}$. It can be noticed that clocks c and c_1 are placed in a regular manner on the support clock c': their relative position is such that c_1 has been obtained through an $(9, 6, 9)$-affine transformation applied to c. By definition, clock c_1 is the result of an (n, φ, d)-affine transformation applied to clock c if it can be obtained from c through steps 1 and 2 as follows: 1. Constructing a new clock c' as the union of c with the set of instants obtained by introducing $n - 1$ *fictive* instants between any two successive instants of c (and $-\varphi$ fictive instants before the first instant of c when φ is negative). 2. Defining the clock c_1 as the set of instants $\{dt + \varphi \mid t \in c'\}$, with $c' = \{t \mid t \in \mathbb{N}\}$ (in other words, counting every d instant, starting with the φ^{th} instant of c', or with the first instant of c' when φ is negative). Clocks c and c_1 are then said to be in an (n, φ, d)-affine relation. The above definition can be expressed in an equivalent form as follows: clocks c and c_1 are in (n, φ, d)-affine relation if there exists a clock c' such that c and c_1 can be respectively expressed using the *affine functions* $\lambda t.(nt + \varphi_1)$ and $\lambda t.(dt + \varphi_2)$, with $\varphi_2 - \varphi_1 = \varphi$, with respect to the time indices of c': $c' = \{t \mid t \in \mathbb{N}\}$, $c = \{nt + \varphi_1 \mid t \in c'\}$, $c_1 = \{dt + \varphi_2 \mid t \in c'\}$.

Properties on affine relations can be exploited in order to verify that clocks are *synchronisable*, that is, their sets of instants can be identified (re-synchronised). Consider (Fig. 2) a SIGNAL program which executes two successive DCT operations at each macro-instant t_i, one on a luminance block of an image, noted $B := Dct(A)$, and the second one on the next block of red chrominance of the same image, described by $D := Dct(C)$.

Each DCT function is expressed in ALPHA at the functional level and further refined according to a particular implementation. The SIGNAL specification is refined accordingly and we obtain the timing diagrams of Fig. 5: the clocks of A and C are synchronous and equal to c, the clocks of B and D are respectively c_1 and c_2, and the clocks c' and c'' describe the instants of the excution of the DCT functions on a potential architecture derived in the ALPHA environment.

In the functional SIGNAL-ALPHA specification, clocks c, c_1 and c_2 were synchronous (see Section 2.1 for details). After refinement of the time indices in the SIGNAL-ALPHA specification, the clocks c_1 and c_2 should be re-synchronised in order to preserve the temporal properties of the whole application. Whether the re-synchronisation of c_1 and c_2 is possible given their relative position as illustrated in Fig. 5, or after further adjustments of their time indices, can be decided based on the properties of the affine relations existing between c, c_1

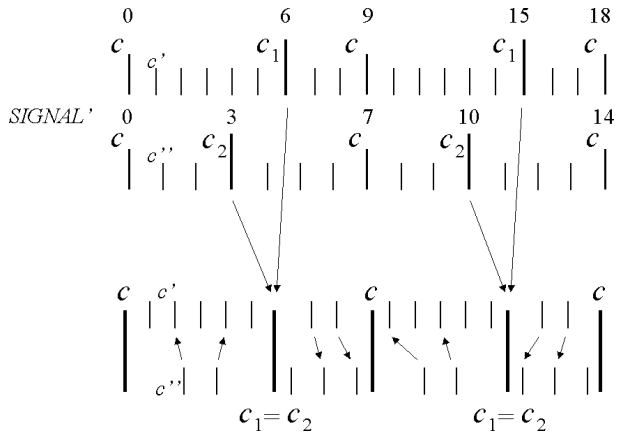

Fig. 5. Synchronisable clocks in the context of codesign with SIGNAL and ALPHA.

and c, c_2 respectively. Clocks c, c_1 and c, c_2 are respectively in $(9, 6, 9)$ and $(7, 3, 7)$-affine relation in the process *SIGNAL'*. The relation existing between the triplets $(9, 6, 9)$ and $(7, 3, 7)$ guarantees the equivalence of the corresponding affine relations. This will be detailed in Section 3. Informally, the equivalence of the above affine relations expresses the fact that the *relative positions* of clocks c and c_1, respectively c and c_2, are identical. Based on this observation, clocks c_1 and c_2 can be identified without contradicting the temporal behaviour of the other clocks in the SIGNAL program. The instants of time of clocks c' and c'' situated between two successive instants of c and c_1 (or c_2) are independent and can be positioned with respect to each other in various manners; in Fig. 5 we have illustrated one possibility. Therefore, c_1 and c_2 can be re-synchronised; we say that c_1 and c_2 are *synchronisable*.

The aim of the affine clock calculus discussed in Section 3 is to define necessary and sufficient conditions for clock synchronisability based on the properties of affine relations on clocks. These conditions are expressed as a set of *synchronisability rules* and are derived in the *semantical model of traces* of SIGNAL. Section 3 begins with an introdution to these concepts.

3 Affine Calculus on Clocks in SIGNAL

Figure 6 introduces the reader to the semantics of traces [12] [16] of SIGNAL. The most important concepts in SIGNAL are: 1. the *signal*, which denotes a variable of the language and represents a finite or infinite sequence of values; 2. the *clock*, a variable associated with each signal which represents the set of logical instants where the values of the signal are present. SIGNAL operators manipulate signals by imposing implicit or explicit constraints on their values and clocks. Constraints on clocks are usually expressed as identities between

clock expressions constructed using the operators of intersection ($\wedge$), union ($\vee$) or difference ($\backslash$). Clocks can be also subsets of other clocks defined as samplings by boolean conditions. When no condition is explicitly or implicitly stated on a pair of clocks, they are independent.

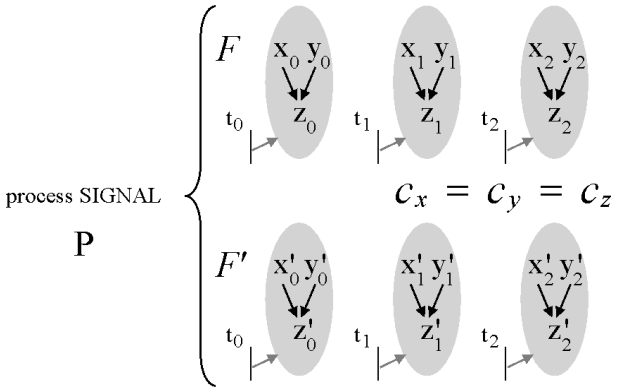

Fig. 6. Illustration of SIGNAL semantics of traces.

A SIGNAL program describes a real-time system, which is in continuous interaction with its environment. Input values are transformed corresponding to the actions of a given specification and the results are provided to the environment. This situation is illustrated in Fig. 6 in the case of a program manipulating inputs x and y and providing output z depending on the values of x and y. In case z is the addition of x and y, signals x, y and z are implicitly constrained by the $+$ operator in SIGNAL to have the same clocks $c_x = c_y = c_z$.

The configurations F and F' illustrated in Fig. 6 correspond to two different executions of the SIGNAL program, involving sequences x_i, y_i and z_i and respectively x'_i, y'_i and z'_i. The set of all possible configurations, called *traces*, which can be exhibited during the execution of a SIGNAL program, defines completely the *process* P associated with the program. Consider A a subset of the set B of signals manipulated by a program. A trace may contain instants with no action involving signals from A. However, each instant of this type contains actions which involve other signals from the set $B \backslash A$. Given a subset A of signals, a *flow* on A is a trace with at least one action involving signals from A for each logical instant. In the particular case of Fig. 6, if we consider the subset of signals to be $\{x, y, z\}$, the traces illustrated are actually flows.

More generally, the process P associated with a SIGNAL program is a set of flows on the variables of the program. Each flow F in P is constrained by a system of equations on the clocks and values of signals manipulated by P. Equations on values can be further expressed in the abstract form of a data dependency graph (an example of a data dependency graph is illustrated in Fig. 6 for the $+$

operator). Besides the clock calculus, the compiler verifies data consistency by checking the absence of cycles in the data dependency graph. In the next section however, we will concentrate mainly on the clock calculus.

3.1 Clock calculus & Synchronisability

The clock calculus is equivalent to the resolution of a system of clock equations. For example:

$$\begin{array}{l} c = c_1 \\ c' = (c_1 \wedge c_2) \vee c_1 \\ c = c' \end{array} \tag{1}$$

can be a system derived from a SIGNAL program which manipulates clocks c, c', c_1 and c_2. In this simple system, c_1 and $(c_1 \wedge c_2) \vee c_1$ have clearly to be proved equivalent, which is an immediate consequence of the axioms of the *boolean lattice*. The space of clocks associated with a SIGNAL program is a boolean lattice [6] the properties of which are extensively used for the proof of equivalences. The resolution of the system is performed by *triangularisation* of the system [5] [1].

Given a boolean signal Cd, its clock, denoted $\hat{Cd}$, can be partitioned into the clock $[Cd]$ where the signal Cd is present and true and the clock $[\neg Cd]$ where Cd is present and false (the clocks $[Cd]$ and $[\neg Cd]$ represent *samplings by boolean conditions*). The relations between clocks $\hat{Cd}$, $[Cd]$ and $[\neg Cd]$ are expressed by the *partition equations* below:

$$\begin{array}{l} [Cd] \vee [\neg Cd] = \hat{Cd} \\ [Cd] \wedge [\neg Cd] = \emptyset \end{array} \tag{2}$$

The axioms of the boolean lattice together with the partition equations induce on the space of clocks a lattice of an order $\preceq$ "coarser" than the order $\leq$ of the boolean lattice [5]. Clocks can be boolean formulas constructed either with samplings by boolean conditions $[Cd]$, $[\neg Cd]$ or with free variables of the boolean lattice. The properties of the lattice of order $\preceq$ are actually used during the triangularisation of any system of clock equations.

The axioms of the lattice $\preceq$ represent a system of *synchronisability rules* in the sense described below. Clocks c and c' are *synchronisable* in the process P, which is denoted by $c \overset{P}{\odot} c'$, if there *exists* a flow F in P in which c and c' are *synchronous*:

$$c \overset{P}{\odot} c' \Leftrightarrow \exists F \in P, c \overset{F}{=} c' \tag{3}$$

(we note $c \overset{F}{=} c'$ the fact that c and c' are synchronous in F).

Whenever the property expressed by equation 3 is valid *for each* flow F in P, the clocks c and c' are said to be *synchronous* in P, which is denoted by $c \overset{P}{=} c'$. This definition can be expressed as follows:

$$c \overset{P}{=} c' \Leftrightarrow \forall F \in P, c \overset{F}{=} c' \tag{4}$$

Unless explicitly constrained through the SIGNAL program, clocks c and c' are completely independent in the associated P process. Therefore, their relative position can be such that in some flows F in P they are identical, while in some other flows F' in P their instants interleave in an arbitrary manner: obviously, if c and c' are independent in P, they are synchronisable. When the relative position of clocks c and c' is implicitly or explicitly constrained by the SIGNAL operators, flows F in P are subsequently constrained and the synchronisability of c and c' depends on these constraints.

In order to better understand the use of the synchronisability rules, consider for example a process P derived from a SIGNAL program *Prg* in which clocks c and c' are defined by the first two equations of the system (1):

$$\begin{aligned} c &= c_1 \\ c' &= (c_1 \wedge c_2) \vee c_1 \end{aligned} \tag{5}$$

Program *Prg* may be transformed into *Prg'* in which an additional constraint has been expressed on clocks c and c': $c = c'$ (in the SIGNAL-ALPHA context, *Prg* could be part of a transformed SIGNAL-ALPHA specification, as seen above, and *Prg'* the same specification, in which clocks are resynchronised). Consider the process P' corresponding to the program *Prg'*. The system of clock equations associated with *Prg'* is (1). Given the set of flows $\mathcal{F}' \subseteq P$ such that $c \overset{F}{=} c'$, $\forall F \in \mathcal{F}'$, it results $P' = \mathcal{F}'$. Therefore, verifying the consistency of (1), which is equivalent to testing that clocks c and c' are equivalent in P', is further equivalent to testing the synchronisability of c and c' in P. The rule $(c_1 \wedge c_2) \vee c_1 = c_1$ from the boolean lattice is indeed a *synchronism rule*: $(c_1 \wedge c_2) \vee c_1 \overset{P}{=} c_1$ for every process P. The same axiom holds for the process P associated with *Prg*. And thus $(c_1 \wedge c_2) \vee c_1 \overset{P}{\odot} c_1$, since synchronism implies synchronisability. Therefore in the example, $\mathcal{F}'$ is not empty and it can be concluded that P' is consistent from the point of view of the constraints expressed on its clocks.

The rules of the lattice $\preceq$ represent synchronisability rules: each identity $f_1 = f_2$, with f_1, f_2 boolean formulas on clocks, is equivalent to $f_1 \overset{P}{=} f_2$ which implies $f_1 \overset{P}{\odot} f_2$ for every process P. These rules can be further extended using the properties of the affine relations between clocks. Figure 5 illustrates this idea: if P is the process associated with the program *SIGNAL'*, the configuration in which clocks c_1 and c_2 coincide represent a flow $F \in P$ such that $c_1 \overset{F}{=} c_2$. Thus, c_1 and c_2 are synchronisable in P. The reason here is that the $(9, 6, 9)$ and $(7, 3, 7)$-affine relations existing respectively between c, c_1 and c, c_2 are equivalent. In the next section, we define the affine relation associated with a flow and a process and further explicitate the concept of equivalence of affine relations.

3.2 Affine Relations in SIGNAL

Given $n, d \in \mathbb{N}^*$ and $\varphi \in \mathbb{Z}$ fixed, clocks c and c_1 are in (n, φ, d)-affine relation in the flow F—which is denoted $c \, \mathcal{R}^F_{(n,\varphi,d)} \, c_1$ or $(c, c_1) \in \mathcal{R}^F_{(n,\varphi,d)}$—if the relative

position of c and c_1 in F can be induced by an (n, φ, d)-affine transformation as defined in Section 2.2.

Clocks c and c_1 are in (n, φ, d)-affine relation in process P, denoted $c\ \mathcal{R}^P_{(n,\varphi,d)}\ c_1$ or $(c, c_1) \in \mathcal{R}^P_{(n,\varphi,d)}$, if they are in (n, φ, d)-affine relation in each flow F of P, i.e. $c\ \mathcal{R}^F_{(n,\varphi,d)}\ c_1$, $\forall F \in P$. Flows and processes are defined over the set of variables they manipulate. For a given set A, a flow F on A is a member of the set of flows $\mathcal{F}_A$ that can be constructed with the variables of A. In a similar manner, a process P on A belongs to the set of processes on A, i.e. $P \in \mathcal{P}_A$. Because of the finite nature of the sets of variables associated with flows and processes, affine relations can be defined as finite sets as follows:

$$\forall F \in \mathcal{F}_A,\ \mathcal{R}^F_{(n,\varphi,d)} = \{(c, c_1) \in A \times A \mid c\ \mathcal{R}^F_{(n,\varphi,d)}\ c_1\} \tag{6}$$

$$\forall P \in \mathcal{F}_A,\ \mathcal{R}^P_{(n,\varphi,d)} = \{(c, c_1) \in A \times A \mid c\ \mathcal{R}^P_{(n,\varphi,d)}\ c_1\} \tag{7}$$

Consider the process $P \in \mathcal{P}_{\{c,c_1,c_2\}}$ defined as follows:

$$P = \{F \in \mathcal{F}_{\{c,c_1,c_2\}} \mid c\ \mathcal{R}^F_{(n_1,\varphi_1,d_1)}\ c_1, c\ \mathcal{R}^F_{(n_2,\varphi_2,d_2)}\ c_2\} \tag{8}$$

(induced by a SIGNAL program that manipulates only the clocks c, c_1 and c_2).

From the definition of an affine relation associated with a process it results $c\ \mathcal{R}^P_{(n_1,\varphi_1,d_1)}\ c_1$ and $c\ \mathcal{R}^P_{(n_2,\varphi_2,d_2)}\ c_2$. Clocks c_1 and c_2 are synchronisable in P if there exists $F \in P$ satisfying $c_1 \overset{F}{=} c_2$. Consider $F_s \in P$ satisfying $c_1 \overset{F_s}{=} c_2$. Obviously $c\ \mathcal{R}^{F_s}_{(n_1,\varphi_1,d_1)}\ c_1$ and $c\ \mathcal{R}^{F_s}_{(n_2,\varphi_2,d_2)}\ c_2$. Being identical in F_s, clocks c_1 and c_2 can be replaced with each other and therefore $c\ \mathcal{R}^{F_s}_{(n_1,\varphi_1,d_1)}\ c_1$ implies $c\ \mathcal{R}^{F_s}_{(n_1,\varphi_1,d_1)}\ c_2$ and $c\ \mathcal{R}^{F_s}_{(n_2,\varphi_2,d_2)}\ c_2$ implies $c\ \mathcal{R}^{F_s}_{(n_2,\varphi_2,d_2)}\ c_1$. It results therefore that $\mathcal{R}^{F_s}_{(n_1,\varphi_1,d_1)} = \mathcal{R}^{F_s}_{(n_2,\varphi_2,d_2)} = \{(c, c_1), (c, c_2)\}$. In conclusion, a necessary condition for clocks c_1 and c_2 to be synchronisable in P is that $\mathcal{R}^{F_s}_{(n_1,\varphi_1,d_1)}$ and $\mathcal{R}^{F_s}_{(n_2,\varphi_2,d_2)}$ be *equivalent*. In the case of the process P defined by (8), it can be proved that this condition is also sufficient.

The equivalence of affine relations depends on the closure properties of the space of affine relations with respect to the main operations that can be applied to it. These are either union, intersection or difference induced by the homonym operations on clocks, or general operations on relations like inverse and composition [15]. In the next section we propose a study of these properties in the semantical model of traces of SIGNAL.

3.3 Properties on Affine Relations & Synchronisability Rules

The semantics of traces. Consider a finite set of signals A. The set of all possible flows defined on A is denoted $\mathcal{F}_A$. Subsets of flows from $\mathcal{F}_A$ can be grouped in processes which are members of the set $\mathcal{P}_A$ of all processes that can be defined on A. A SIGNAL program on A defines a process $P \in \mathcal{P}_A$; each flow

$F \in P$ satisfies some constraints imposed by the SIGNAL operators on the clocks and values of the signals from A.

SIGNAL disposes of four basic operators (kernel) which are sufficient for the construction of any program regardless of its complexity. Kernel operators are combined through composition and restriction in order to build programs. The composition and restriction of programs induce naturally the corresponding operations on processes and flows. Intuitively, the restriction of a flow F to a set of variables $A' \subseteq A$ is the flow $\Pi_{A'}(F)$ which contains only those instants of F with actions involving signals from A'.

Concerning processes, the main operations are defined as follows. Given a set of variables $A' \subseteq A$, the restriction of $P \in \mathcal{P}_A$ to A' (the projection of P on A') contains the flows $F \in P$ manipulating exclusively variables of A':

$$\Pi_{A'}(P) = \{F' \in \mathcal{F}_{A'} \mid F' = \Pi_{A'}(F), \forall F \in P\} \tag{9}$$

The composition of processes $P_1 \in \mathcal{P}_{A_1}$ and $P_2 \in \mathcal{P}_{A_2}$, with A_1, A_2 arbitrary sets of variables, is defined by:

$$P_1 \mid P_2 = \{F \in \mathcal{F}_{A_1 \cup A_2} \mid \Pi_{A_1}(F) \in P_1, \Pi_{A_2}(F) \in P_2\} \tag{10}$$

The following lemma describes the necessary and sufficient conditions—stated as $\Pi_{A_2}(P) \subseteq Q$—for a property valid in the process Q to be also also in P:

Lemma 1. $\forall P \in \mathcal{P}_{A_1}$, $\forall Q \in \mathcal{P}_{A_2}$, $A_2 \subseteq A_1$,

$$\Pi_{A_2}(P) \subseteq Q \Leftrightarrow P \mid Q = P \tag{11}$$

In other words, given the hypothesis described by the left hand side of (11), Q expresses a property valid also in P.

Properties on affine relations. Operations specific to relations in general, like inverse $()^{-1}$ and composition $*$, can be applied to affine relations [15]. As an example, consider a process $P \in \mathcal{P}_{\{c,c_1,c_2,c_3\}}$ with clocks c, c_1, c_2 and c_3 satisfying $c\ \mathcal{R}^P_{(n_1,\varphi_1,d_1)}\ c_1$, $c_1\ \mathcal{R}^P_{(n_2,\varphi_2,d_2)}\ c_2$ and $c\ \mathcal{R}^P_{(n_3,\varphi_3,d_3)}\ c_3$. Obviously, it results that $c\ \mathcal{R}^P_{(n_1,\varphi_1,d_1)}\ *\ \mathcal{R}^P_{(n_2,\varphi_2,d_2)}\ c_2$ and the synchronisability of c_2 and c_3 depends on properties of the composition. When the space of affine relations is closed under composition, the test of the synchronisability of c_2 and c_3 reduces itself to the verification of the equivalence of affine relations.

Affine relations can be further combined through union $\cup_r$, intersection $\cap_r$ and difference $\setminus_r$ induced by the homonym operations on clocks ($\vee$, $\wedge$, $\setminus$). A similar argument as before conducts to the necessity of studying closure properties of these operators with respect to the space of affine relations.

Here is a brief presentation of the main steps and results obtained in the study of affine relations.

Equivalence of Affine Relations. An equivalence relation, noted $\sim$, can be defined between triplets (n, φ, d) as follows: $(n, \varphi, d) \sim (n', \varphi', d')$ iff either $nd' = n'd$ and $n\varphi' = n'\varphi$, for $G \mid \varphi$ (i.e., G is a divisor of φ) and $G' \mid \varphi'$, or $nd' = n'd$ and $\left[\frac{dt+\varphi}{n}\right] = \left[\frac{d't+\varphi'}{n'}\right], \forall t \in \mathbb{N}, dt+\varphi \geq 0$, for $G \not| \varphi$ and $G' \not| \varphi'$, with $G = gcd(n, d)$ the greatest common divisor of n and d, $G' = gcd(n', d')$ and $[x]$ the integer part of $x \in \mathbb{N}$. The equivalence of affine relations depends exclusively on the values of the associated triplets (n, φ, d) [17]:

Proposition 1.

$$\mathcal{R}^F_{(n,\varphi,d)} = \mathcal{R}^F_{(n',\varphi',d')}, \ \forall \ F \in \mathcal{F}_A \Leftrightarrow (n, \varphi, d) \ \sim \ (n', \varphi', d') \tag{12}$$

Canonical Form. In order to reduce the complexity of the test of the equivalence $\sim$, we have then defined a canonical form $(n_{CF}, \varphi_{CF}, d_{CF})$ for a triplet (n, φ, d) [18] as follows:

Proposition 2.

$$\begin{array}{ll} a) & G \mid \varphi \Rightarrow (n_{CF}, \varphi_{CF}, d_{CF}) = (\frac{n}{G}, \frac{\varphi}{G}, \frac{d}{G}) \\ b) & G \not| \varphi \Rightarrow (n_{CF}, \varphi_{CF}, d_{CF}) = (2\frac{n}{G}, (2\left[\frac{\varphi}{G}\right] + 1), 2\frac{d}{G}) \end{array} \tag{13}$$

Consequently, the canonical form of $\mathcal{R}^F_{(n,\varphi,d)}$ is $\mathcal{R}^F_{(n_{CF},\varphi_{CF},d_{CF})}$ and the verification of the identity of two affine relations is thus reduced to the verification that two triplets of integers are identical:

Proposition 3.

$$\mathcal{R}^F_{(n,\varphi,d)} = \mathcal{R}^F_{(n',\varphi',d')} \Leftrightarrow (n_{CF}, \varphi_{CF}, d_{CF}) = (n'_{CF}, \varphi'_{CF}, d'_{CF}) \tag{14}$$

Operations on affine relations. If any expression on affine relations could be rewritten as an affine relation, the verification of clock synchronisability would consist only in a test of equivalence on affine relations as above. But it has been observed that this was not the case in general. The closure property is true for the inverse of an affine relation. Also, the affine relation $\mathcal{R}^F_{(1,0,1)}$ is neutral with respect to composition. However, the closure property is lost when dealing with composition. The composition of two general affine relations $\mathcal{R}^F_{(n,\varphi,d)}$ and $\mathcal{R}^F_{(n',\varphi',d')}$ does not generally produce an affine relation. Nevertheless, it has been possible to identify in the space of the affine relations $\mathcal{R}^F_{(n,\varphi,d)}$ a subspace consisting of relations of the form $\mathcal{R}^F_{(1,\varphi,d)}$, with $\varphi \geq 0$, in which the closure property is true. Following this observation, we have distinguished two cases, as detailed in the sequel.

Properties of affine relations $\mathcal{R}^F_{(1,\varphi,d)}$, with $\varphi \geq 0$. It has been demonstrated [16] that the space of affine relations $\mathcal{R}^F_{(1,\varphi,d)}$, although closed under composition $*$ and intersection $\cap_r$, is not closed under union $\cup_r$ and difference $\setminus_r$. It is therefore necessary to define necessary and sufficient conditions for the equivalence

of arbitrary expressions constructed with affine relations of the form $\mathcal{R}^F_{(1,\varphi,d)}$ using composition, union, intersection and difference. Given the complexity of the space of expressions on affine relations $\mathcal{R}^F_{(1,\varphi,d)}$ and the necessity of efficient algorithms for testing their equivalence, the question of the existence of a canonical form appears. Our attempt to provide a canonical form using exclusively the $\cup_r$ operator—based on the observation that any expression in this space can be rewritten as a union of affine relations $\mathcal{R}^F_{(1,\varphi,d)}$—has failed because of the infinite number of possibilities in which a relation $\mathcal{R}^F_{(1,\varphi,d)}$ can be rewritten as a union of affine relations of the same type. However, in [16] we propose a *relative normal form* which reduces partially the complexity of the equivalence calculus.

Properties of general affine relations $\mathcal{R}^F_{(n,\varphi,d)}$. Deciding that two arbitrary expressions on general affine relations are equivalent is a difficult problem. An initial step may be to isolate subsets of triplets (n,φ,d) and (n',φ',d') which respect the condition that the result of the operation $\mathcal{R}^F_{(n,\varphi,d)}\ op_r\ \mathcal{R}^F_{(n',\varphi',d')}$, with $op_r \in \{*, \cup_r, \cap_r, \setminus_r\}$, is an affine relation. In [16] we propose a subset of such triplets $\{(n,\varphi,d),(n',\varphi',d')\}$, for which the above property is true, for the composition. Computing this subset $\{(n,\varphi,d),(n',\varphi',d')\}$ is an NP-complete problem. Future work may consider the applicability of heuristic search methods for this computation. Another open problem is the study of the properties of the union $\cup_r$, intersection $\cap_r$ and difference $\setminus_r$ of general affine relations.

Synchronisability rules. The main results concerning the particular affine relations $\mathcal{R}^F_{(1,\varphi,d)}$, with $\varphi \geq 0$, and the general ones $\mathcal{R}^F_{(n,\varphi,d)}$ have respectively permitted the induction of a set of synchronism rules and a set of synchronisability rules. These rules actually represent a set of conditions which are necessary and sufficient for the synchronism and respectively the synchronisability of two clocks.

An example of synchronism rule is given below. Consider the process $P \in \mathcal{P}_{\{c,c_1,c_2,c_3\}}$ defined by:

$$P = \{F \in \mathcal{F}_{\{c,c_1,c_2,c_3\}} \mid c\ \mathcal{R}^F_{(1,\varphi_1,d_1)}\ c_1, c_1\ \mathcal{R}^F_{(1,\varphi_2,d_2)}\ c_2, c\ \mathcal{R}^F_{(1,\varphi_3,d_3)}\ c_3\} \quad (15)$$

Obviously $c\ \mathcal{R}^P_{(1,\varphi_1,d_1)}\ c_1$, $c_1\ \mathcal{R}^P_{(1,\varphi_2,d_2)}\ c_2$ and $c\ \mathcal{R}^P_{(1,\varphi_3,d_3)}\ c_3$. The calculus on affine relations $\mathcal{R}^F_{(1,\varphi,d)}$ induces $\mathcal{R}^F_{(1,\varphi_1,d_1)} * \mathcal{R}^F_{(1,\varphi_2,d_2)} = \mathcal{R}^F_{(1,\varphi_1+d_1\varphi_2,d_1d_2)}$ which is valid also for processes: $\mathcal{R}^P_{(1,\varphi_1,d_1)} * \mathcal{R}^P_{(1,\varphi_2,d_2)} = \mathcal{R}^P_{(1,\varphi_1+d_1\varphi_2,d_1d_2)}$. Therefore $c\ \mathcal{R}^P_{(1,\varphi_1+d_1\varphi_2,d_1d_2)}\ c_2$, and c_2 and c_3 are synchronisable if and only if $\mathcal{R}^P_{(1,\varphi_1+d_1\varphi_2,d_1d_2)} = \mathcal{R}^P_{(1,\varphi_3,d_3)}$. With Propositions 2 and 3, $\mathcal{R}^P_{(1,\varphi_1+d_1\varphi_2,d_1d_2)}$ and $\mathcal{R}^P_{(1,\varphi_3,d_3)}$ are equivalent if and only if $(1,\varphi_1+d_1\varphi_2,d_1d_2)$ and $(1,\varphi_3,d_3)$ are identical, that is, $\varphi_1+d_1\varphi_2=\varphi_3$ and $d_1d_2=d_3$. This result is expressed in the following synchronism rule:

Proposition 4. *$\forall P \in \mathcal{P}_{\{c,c_1,c_2,c_3\}}$ with c, c_1, c_2 and c_3 satisfying $c\ \mathcal{R}^P_{(1,\varphi_1,d_1)}\ c_1$, $c_1\ \mathcal{R}^P_{(1,\varphi_2,d_2)}\ c_2$ and $c\ \mathcal{R}^P_{(1,\varphi_3,d_3)}\ c_3$, the following equivalences are*

verified:

$$c_2 \overset{P}{\odot} c_3 \Leftrightarrow \left\{ \begin{array}{l} \varphi_1 + d_1\varphi_2 = \varphi_3 \\ d_1 d_2 = d_3 \end{array} \right\} \Leftrightarrow \; c_2 \overset{P}{=} c_3 \tag{16}$$

In Fig. 7 the particular case $\varphi_1 = 6$, $d_1 = 2$, $\varphi_2 = 1$, $d_2 = 2$, and $\varphi_3 = 8$, $d_3 = 4$ is illustrated. It can be observed that clock c_1 is an *affine sampling* of phase φ_1 and period d_1 on clock c. Clock c_2 is defined similarly by an affine sampling of parameters φ_2 and d_2 on c_1. The same clock c_2 can be obtained by an affine sampling of φ_3 and d_3 on c; the clock c_3 constructed in this manner is synchronous, and therefore synchronisable, with c_2.

Following a sequence of steps similar as for Proposition 4, we have derived a system of synchronism rules which is minimal; it enables the verification of the synchronisability of two arbitrary clocks related by an expression on affine relations $\mathcal{R}^F_{(1,\varphi,d)}$, with $\varphi \geq 0$. The results concerning the equivalence of general affine relations $\mathcal{R}^F_{(n,\varphi,d)}$, summarized by Propositions 1, 2 and 3, and the partial result on composition of general affine relations, have allowed the derivation of a set of synchronisability rules which are sufficient for the validation of SIGNAL programs for which the single operation performed on affine relations is composition. Further work should be dedicated to the study of the union $\cup_r$, intersection $\cap_r$ and difference $\setminus_r$ of general affine relations.

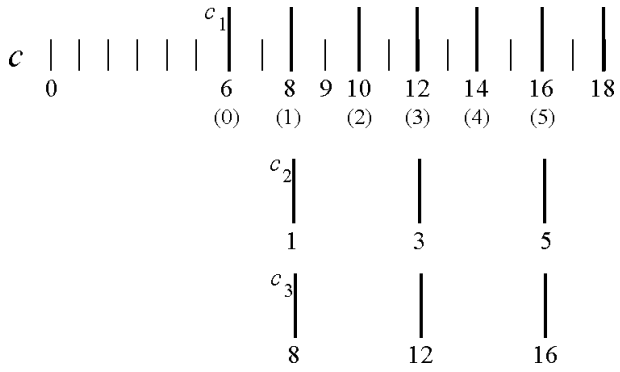

Fig. 7. Illustration of Proposition 4.

3.4 Implementation of the Affine Clock Calculus

A prototype implementing the synchronisability rules introduced in Section 3.3 has been integrated with the existing clock calculus and used for the validation of the SIGNAL-ALPHA interface on the video image coding application introduced in Section 2. In Section 3.1 we have explained that the existing (boolean)

clock calculus relies on the properties of the lattice $\preceq$ existing on the space of clocks, and that it is equivalent to a system of synchronisability rules. The implementation of the affine clock calculus is briefly described now. By choosing an appropriate implementation of a general affine relation $\mathcal{R}^P_{(n,\varphi,d)}$ as detailed in [16], the considered clock expressions contain formulas constructed only with *affine clocks*, that is, affine samplings of specified phase and period on a given basis clock. Thus, the order $\preceq_{aff}$ defined by

$$\preceq_{aff} = \{(c_1, c_2) | \ \exists \varphi_i \geq 0, d_i > 1, \mathcal{R}^P_t = \mathcal{EXP}(\ldots, \mathcal{R}^P_{(1,\varphi_i,d_i)}, \ldots), c_1 \mathcal{R}^P_t c_2\} \quad (17)$$

with $\mathcal{EXP}$ a general expression on affine relations, induces on the space of *affine clocks* a lattice structure. The system of equations on affine clocks associated with a SIGNAL program is solved by triangularisation. When the equivalence of two clock expressions has to be demonstrated, synchronisability rules such that deduced in Section 3.3 are applied. Finally, for the integration of the affine and boolean clock calculus, each synchronisability rule which has been deduced in a process $Q \in \mathcal{P}_{A_2}$, is used in a larger context $P \in \mathcal{P}_{A_1}$, with $A_2 \subseteq A_1$, satisfying $\Pi_{A_2}(P) \subseteq Q$. Following Lemma 1, the synchronisability rule is also valid in P.

4 Application

The affine clock calculus has been used for the validation of the video image coding application described in Section 2. This application contains an important control part, which has been programmed in SIGNAL, and operations like the DCT, which have been expressed in ALPHA. The application has been specified and simulated at both functional and architectural levels as described in Section 2. In the coding system described in [8], each image is decomposed into a fixed number of macro-blocks, each macro-block consisting of one block of luminance and two blocks of chrominance (red and blue). At the architectural level, we have refined the ALPHA specifications of the DCTs corresponding to the blocks of luminance and red chrominance of a macro-block. These temporal refinements have been expressed in SIGNAL by means of two general affine relations between clocks c, c_1 and c, c_2 as illustrated in Fig. 5. The synchronisability of c_1 and c_2 has been verified by compilation and the entire SIGNAL-ALPHA system has been simulated in C.

Most of the operations involved in image coding applications are critical from the point of view of execution time or resources. Therefore, a codesign approach can be considered. The affine clock calculus represents an important element in defining a complete codesign methodology based on the SIGNAL and ALPHA languages. Besides the cospecification and cosimulation of an application, using SIGNAL and ALPHA in a codesign framework is interesting since it offers solutions to other codesign problems such as the automatic synthesis of specialised circuits for regular algorithms, or the generation of optimal code for the software implementation of both calculations and control. Concerning the latter, one might consider the hardware/software partitioning of an application corresponding to the partitioning into SIGNAL and ALPHA subsystems. Therefore,

ALPHA processes would be implemented in hardware by automatic synthesis, while SIGNAL processes would be translated into C code for general purpose architectures. However, the proposed partitioning is not unique and automatic hardware/software partitioning remains an open problem, as it is the implementation of the hardware/software interface.

5 Conclusion

The joint use of the SIGNAL and ALPHA languages in hardware/software codesign has introduced the problem of the validation of mixed SIGNAL-ALPHA specifications both at the functional and architectural levels. The refinement of SIGNAL-ALPHA specifications towards the architectural level and their subsequent validation necessitates the extension of the formal clock calculus implemented in the SIGNAL compiler. This paper presents the new affine clock calculus based on the properties of affine relations induced between clocks by the refinement of SIGNAL-ALPHA specifications. The properties of affine relations are studied in the semantical model of traces of the SIGNAL language, but can be extended to any general model with similar characteristics. Based on this study, a new set of synchronisability rules is defined and integrated with the set already implemented by the existing formal clock calculus.

The affine clock calculus is relevant for the definition and implementation of a codesign methodology using the SIGNAL and ALPHA languages. Techniques for real-time system validation (formal verification, simulation) available in the SIGNAL and ALPHA environments can be used for cospecification and cosimulation. Both environments also have tools for automatic generation of optimal implementations which can be used in a complementary manner for hardware synthesis and/or implementation on general architectures. Further work should be devoted to the complete integration of the SIGNAL and ALPHA languages thus making possible the use of the most adapted formalism and environment for a given application.

References

1. Amagbegnon T., Besnard L., Le Guernic P.: *Arborescent Canonical Form of Boolean Expressions.* INRIA Research Report 2290, IRISA/INRIA - Rennes, France, 1994
2. Amagbegnon T., Le Guernic P., Marchand H., Rutten E.: *The* SIGNAL *dataflow methodology applied to a production cell.* IRISA Research Report 917, IRISA/INRIA - Rennes, France, 1995
3. Belhadj M.: "Using VHDL for Link to Synthesis Tools". *Proceedings of the North Atlantic Test Workshop*, June 1994, Nmes, France
4. Benveniste A., Berry G.: "Real-Time systems design and programming", *Proceedings of the IEEE*, September 1991, **79**, (9)
5. Besnard L.: *Compilation de SIGNAL : horloges, dpendances, environnement*, PhD Thesis, University of Rennes 1, France, September 1992
6. Birkhoff G.: *Lattice Theory*, AMS colloquium publications, 1973

7. De Micheli G.: "Computer-Aided Hardware-Software Codesign", *IEEE Micro*, August 1994, **14**, (4)
8. ETSI (European Telecommunication Standards Institute) *Specification of Component TV codecs 32-45 Mbit/s.* December 1990
9. Gautier T., Le Guernic P., Quinton P., Rajopadhye S., Risset T., Smarandache I.: "Projet CAIRN: conception d'architectures partir de SIGNAL et ALPHA" *CODESIGN Conception conjointe logiciel-matriel*, Eyrolles, Collection Technique et Scientifique des Tlcommunications, 1998
10. Gupta R.K., Coelho C.N., De Micheli G.: "Program Implementation Schemes for Hardware-Software Systems" *Computer*, January 1994, pp. 48-55
11. Kalavade A., Lee E.A.: "A Hardware-Software Codesign Methodology for DSP Applications" *IEEE Design & Test of Computers*, September 1993, **10**, (3), pp. 16-28
12. Le Guernic P., Gautier T.: "Data-Flow to von Neumann: the SIGNAL Approach", *Advanced Topics in Data-Flow Computing*, (Gaudiot J.-L. and Bic L., 1991), pp. 413-438
13. Le Guernic P., Gautier T., Le Borgne M., Le Maire C.: "Programming Real-time Applications with SIGNAL", *Proceedings of the IEEE*, September 1991, **79**, (9), pp. 1321-1336
14. Salinas M.H., Johnson B.W., Aylor J.H.: "Implementation-Independent Model of an Instruction Set Architecture in VHDL" *IEEE Design & Test of Computers*, September 1993, **10**, (3), pp. 42-54
15. Sanderson J.G.: *A Relational Theory of Computing*, Springer Verlag 1980, **80**, Goss G. and Hartmanis J.
16. Smarandache I.: *Transformations affines d'horloges : application au codesign de systèmes temps-réel en utilisant les langages* SIGNAL *et* ALPHA, PhD Thesis, University of Rennes 1, France, October 1998
17. Smarandache I., Le Guernic P.: "Affine Transformations in SIGNAL and Their Applications in the Specification and Validation of Real-Time Systems" *Transformation-Based Reactive Systems Development*, Proceedings of the 4th International AMAST Workshop on Real-Time Systems and Concurrent and Distributed Software, Palma, Spain, LNCS **1231**, Springer Verlag, 1997
18. Smarandache I., Le Guernic P.: *A Canonical Form for Affine Relations in* SIGNAL. INRIA Research Report 3097, IRISA/INRIA - Rennes, France, 1997
19. Thomas D.E., Adams J.K., Schmit H.: "A Model and Methodology for Hardware-Software Codesign" *IEEE Design & Test of Computers*, September 1993, **10**, (3), pp. 6-15
20. Wilde D.: *The* ALPHA *Language.* IRISA Research Report 827, IRISA/INRIA - Rennes, France, 1994
21. Wilde D., Sié O.: *Regular array synthesis using* ALPHA. IRISA Research Report 829, IRISA/INRIA - Rennes, France, 1994